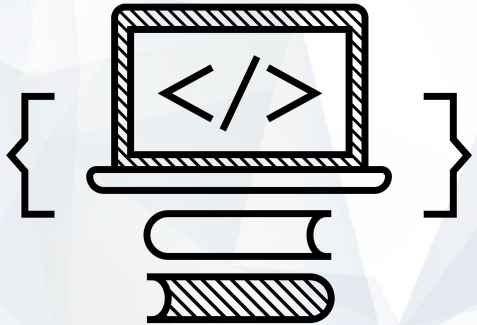
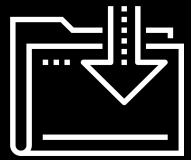


CSS Grids

Skills Bootcamp in Front-End Web Development

Lesson 2.2



WELCOME

Today's Objectives

By the end of class today, you will:

- Design and code basic grids using CSS Grid.
- Build layouts by positioning elements inside grids.
- Create a complex layout by nesting HTML elements inside of a grid.
- Write media queries in code to create a responsive grid layout.

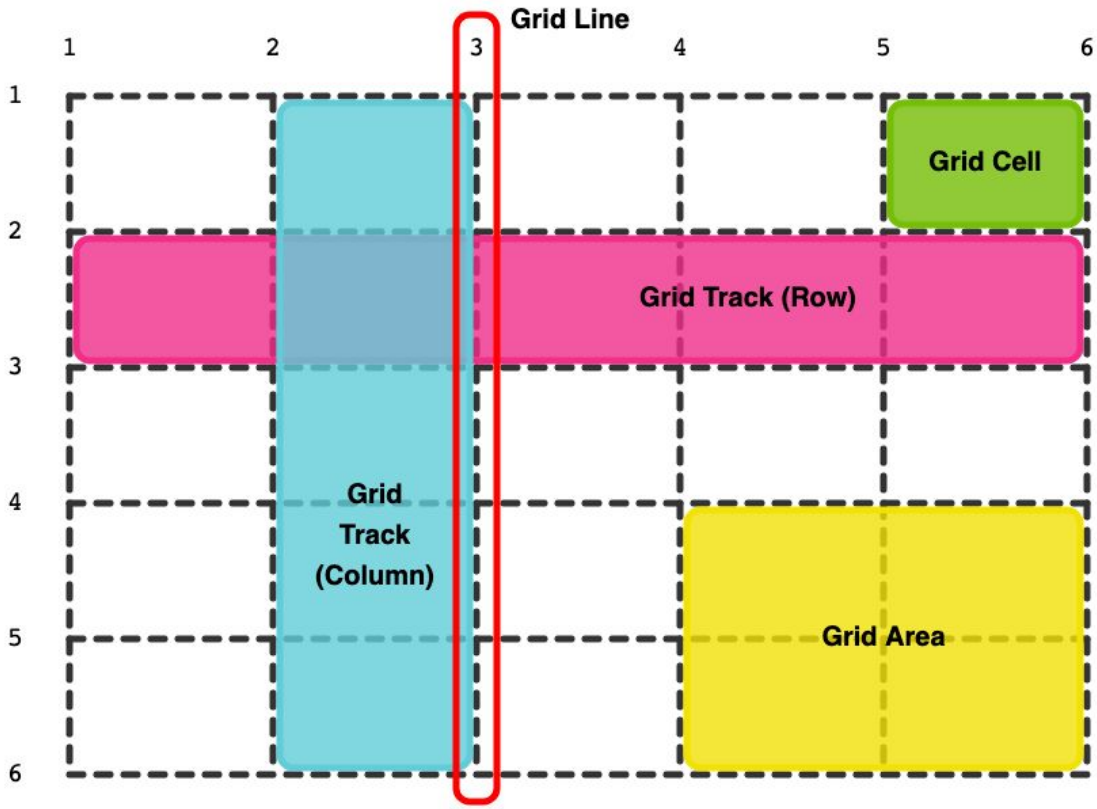

Furniture
New In
Furniture
Rest
Decoration
Outdoor
Magazine
Outlet
New Collection
Chairs & Stools
Armchair upholstered in synthetic leather.
Powder coated steel legs.

Introduction to CSS Grids

Grid Layouts

CSS Grids are used to create two-dimensional layouts that can span multiple columns or rows.

Grids allow you to define the layout section by section, allowing you to create children that span rows or columns and line up next to each other.



https://webkit.org/blog/7434/css-grid-layout-a-new-layout-module-for-the-web/

Before Flex and CSS Grids

Before flex and grids, building layouts was a frustrating and difficult experience.

To build interesting layouts, many CSS hacks were introduced, but the ability to create flexible complex layouts was near impossible, and getting them supported by all browsers was basically impossible.

Before Flex and CSS Grids

In 2013, Microsoft's Edge browser shipped its implementation of the CSS Grid and was quickly adopted by the W3 Consortium (W3C) and soon standardized and supported by all browsers.

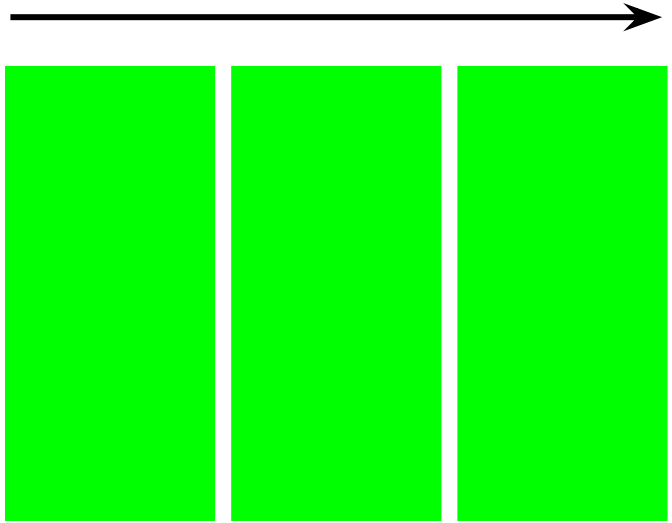
IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Blackberry Browser	Opera Mobile *	Chrome for Android	Firefox for Android	IE Mobile
		2-39	4-28										
		[3] 40-51	[1] 29-56		10-27								
6-9	[2] 12-15	[4] 52-53	[4] 57	3.1-10	[1] 28-43	3.2-10.2							
[2] 10	16-17	54-66	58-74	10.1-12	44-57	10.3-12.1		2.1-4.4.4	7	12-12.1			[2] 10
[2] 11	18	67	75	12.1	58	12.2	all	67	10	46	75	67	[2] 11
	76	68-69	76-78	13-TP		13							

https://caniuse.com/#search=grids

Flexbox vs. CSS Grids

CSS Grids are used to build two-dimensional layouts, meaning you can create full, complex designs as opposed to one-dimensional layouts, which are only left–right or top–bottom.

Flexbox is one-dimensional.



CSS Grids are two-dimensional.

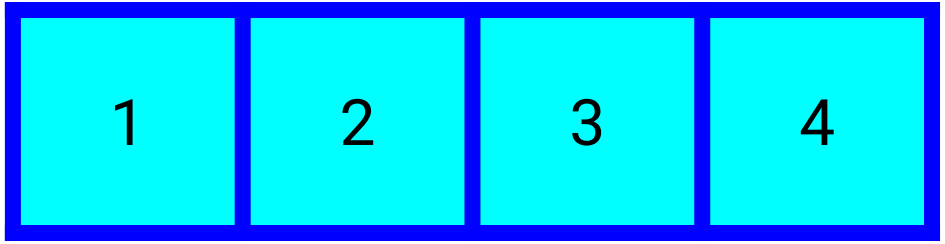
CSS Grid Containers

In order to use a grid, you need a **grid container**.

Grid containers are very similar to flex.

Converting an HTML element into a container is easy. All you need to do is set `display: grid` to any HTML element.

If that element has any children nested inside of it, they become **grid items**.




```
.containerGrid {
  display: grid;
  background-color: #2196F3;
  padding: 10px;
}
```

```
<div class="containerGrid">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

Grid containers contain rows and columns of **grid items**.

CSS Grid Containers

Grids are defined by using two CSS properties:

01

`grid-template-columns` is used to specify the number of columns based on the children you have in your container.

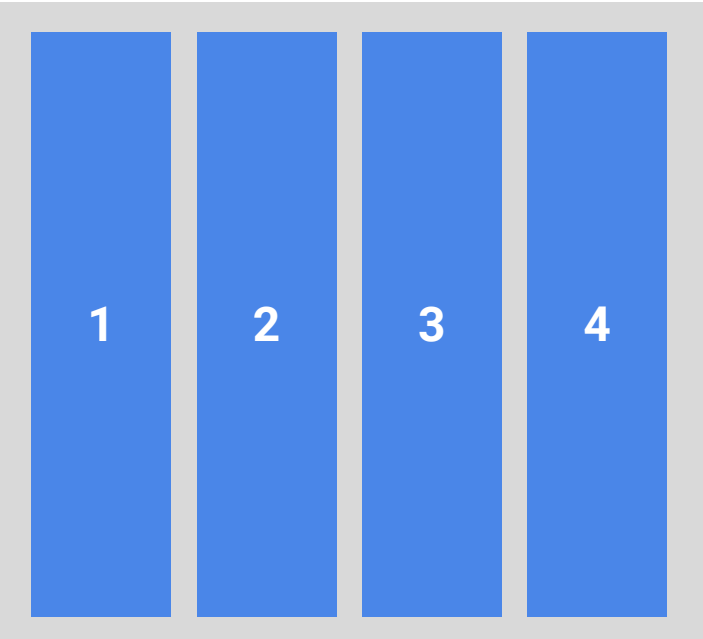
02

`grid-template-rows` is used to specify the number of rows of content that will display in your container.

The `grid-template-columns` Property

CSS Property: `grid-template-columns`

This specifies the number of columns based on the children you have in your container.




```
.containerGrid {
  display: grid;
  grid-template-columns: 25% 25% 25% 25%;
}
```

CSS Property: `grid-template-columns`

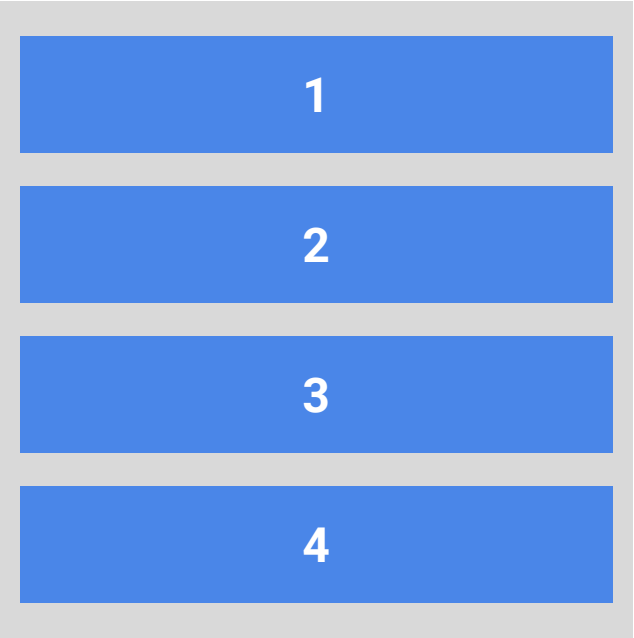
You can customize grid columns to be any size you'd like.


```
.containerGrid {
  display: grid;
  grid-template-columns: 25% 75%;
}
```

The `grid-template-rows` Property

CSS Property: `grid-template-rows`

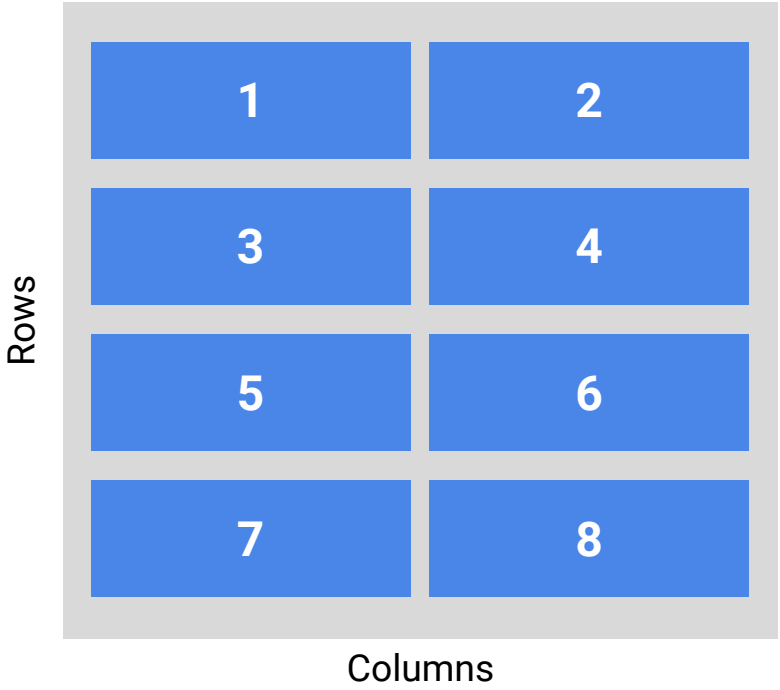
This specifies how many rows (top–bottom) your grid template will have.
With fraction units (more on this later) or percentages, grid children will take the height of their containers.




```
.containerGrid {
  display: grid;
  grid-template-rows: 25% 25% 25% 25%;
}
```

Columns and Rows Create Grids

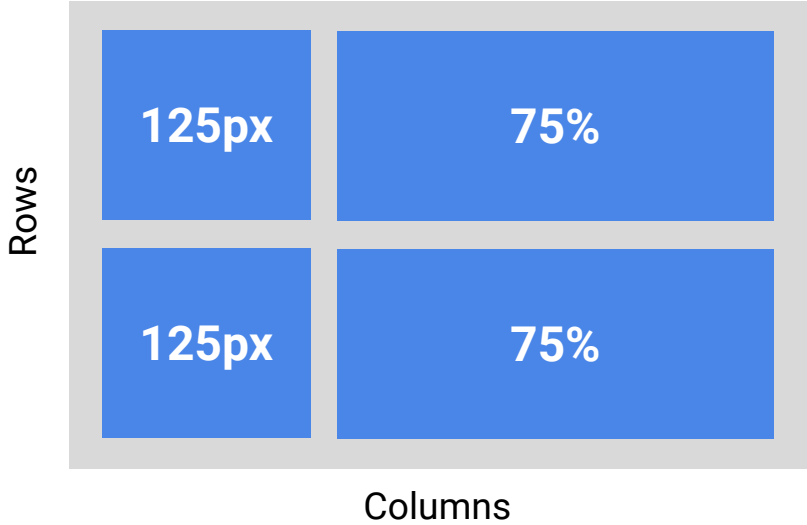
The properties `grid-template-rows` and `grid-template-columns` are used together to create full grid layouts.




```
.containerGrid {
  display: grid;
  grid-template-rows: 25% 25% 25% 25%;
  grid-template-columns: 50% 50%;

}
```

Sizing Grid Areas

CSS Grid areas can be built by using a combination of pixel widths, percentages, or fraction units.



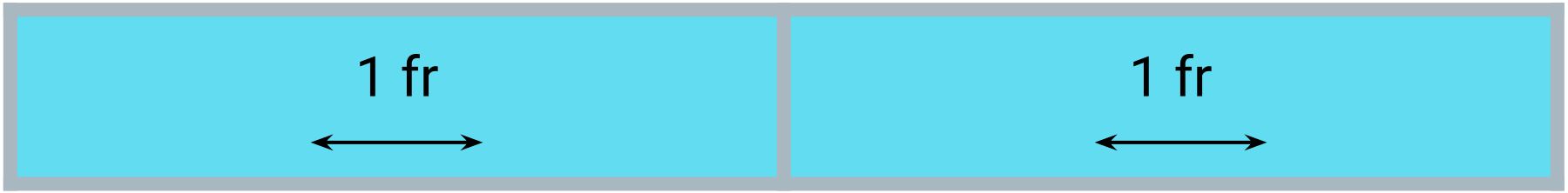

```
.containerGrid {
  display: grid;
  Grid-template-rows:50% 50%;
  grid-template-columns: 125px 75%;

}
```

Fraction units specify the space taken up by each row. Fraction units are calculated with how much space is left in your container and what fraction of the space the element should take up.

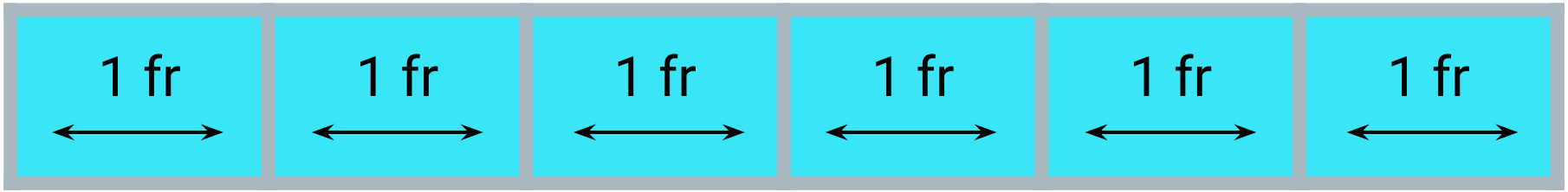
Fraction Units (fr)

Fraction units are calculated with how much space is left in your container and what fraction of the space the element should take up.

Two child elements



Six child elements



The `grid-template-areas` Property

All the properties we just talked about are used to create templates, which are specified by `grid-template-areas`.

Templates are used to display how your content lays out on the page.

CSS Property: `grid-template-areas`

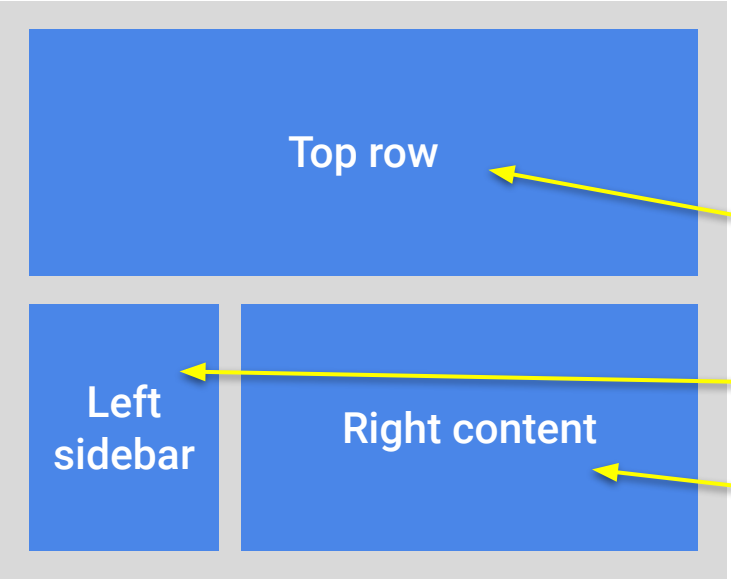
Grid template areas are easy to set up; all you need to do is create a class that targets any individual grid element by using grid-area.


```html
<div class="containerGrid">
  <div class="topRow">topRow</div>
  <div class="leftSidebar">Left Sidebar</div>
  <div class="rightContent">Right Content</div>
</div>
```


```css
.containerGrid {
  display: grid;
  height: 500px;
}
.topRow {
  grid-area: topRow;
}
.leftSidebar {
  grid-area: leftSidebar;
}
.rightContent {
  grid-area: rightContent;
}
```

CSS Property: `grid-template-areas`

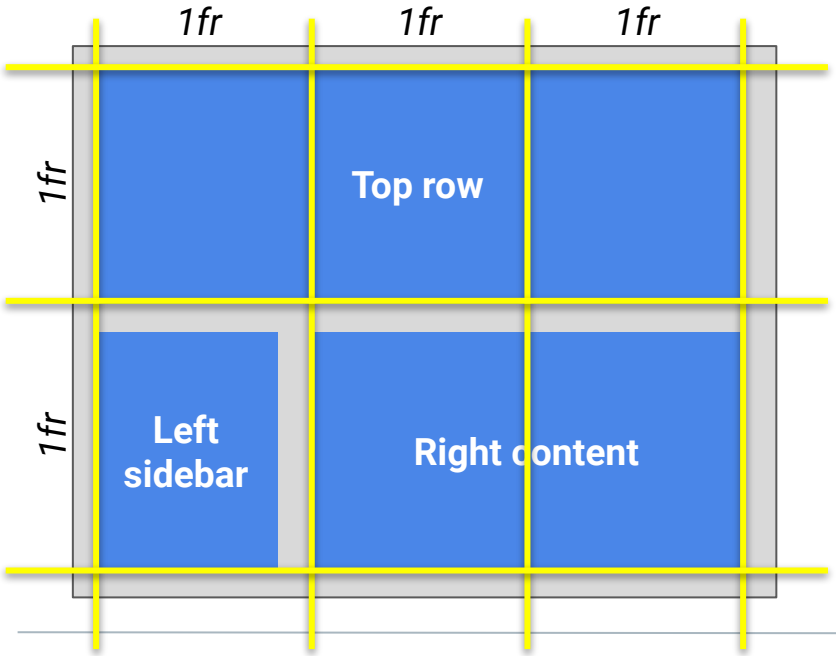
The property grid-template-areas are easy to set up; all you need to do is create a class that targets any individual grid element.




```
.containerGrid {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr;
  grid-template-areas:
    "topRow topRow topRow"
    "leftSidebar rightContent rightContent"
    ;
}
.topRow {
  grid-area: topRow;
}
.leftSidebar {
  grid-area: leftSidebar;
}
.rightContent {
  grid-area: rightContent;
}
```

CSS Property: `grid-template-areas`

Our grid-template-areas provide a visual view (in our code) of how our grid is going to display in relation to the columns and rows we set.




```
.containerGrid {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr;
  grid-template-areas:
    "topRow topRow topRow"
    "leftSidebar rightContent rightContent"
    ;
}
.topRow {
  grid-area: topRow;
}
.leftSidebar {
  grid-area: leftSidebar;
}
.rightContent {
  grid-area: rightContent;
}
```

Instructor Demonstration

grid-template-areas

Activity: CSS Grid, Part 1

In this activity, you'll test out creating grid-based layouts for yourselves.

Suggested Time:

30 minutes

CSS

Time’s Up! Let’s Review.

Let's Review: CSS Grid , Part 1

- What happens to the children of a parent container when you set it to `display: grid`?
- What is a fraction unit in grids?
- Why are grid-template-areas powerful?

Grid Positioning

CSS Grid Positioning: `align-items`

The property `align-content` positions items contained in a grid container along the column axis (top–bottom).

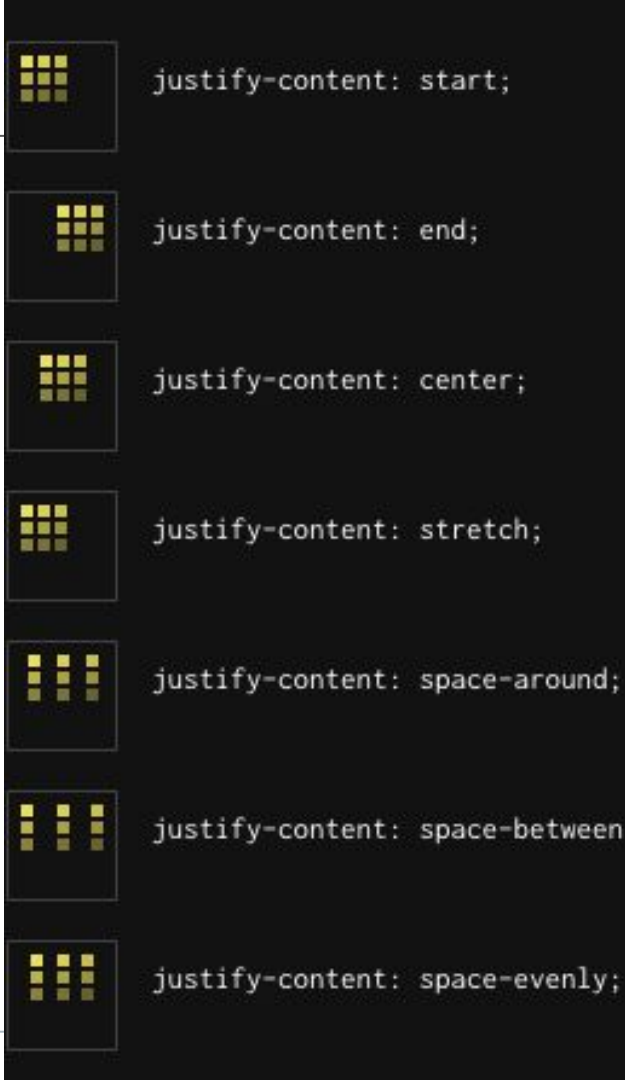
- `start`: Pack items from the start.
- `end`: Pack items from the end.
- `center`: Pack items around the center.
- `stretch`:– Stretch “auto”-sized items to fit the container.
- `space-around`: Lines are evenly distributed in the container with half-size spaces on either end.
- `space-between`: Lines are evenly distributed in the container.
- `space-evenly`: Lines are evenly distributed and centered inside the container.

http://grid.malven.co/

CSS Grid Positioning: `justify-content`

The property `justify-content` is used to justify the grid items along the row axis (left–right).

- `start`: Pack items from the start.
- `end`: Pack items from the end.
- `center`: Pack items around the center.
- `stretch`: Stretch “auto”-sized items to fit the container.
- `space-around`: Lines are evenly distributed in the container with half-size spaces on either end.
- `space-between`: Lines are evenly distributed in the container.
- `space-evenly`: Lines are evenly distributed and centered inside the container.



http://grid.malven.co/

Nesting CSS Grids

Nesting CSS Grids

All layouts are built by nesting HTML elements inside each other. When you nest a grid inside another grid, you can control the position of the child grid, allowing you to create complex layouts.

You do this by declaring a child element of a grid container that it is now `display: grid;`. It will accept all the properties that we have been practicing in the previous examples.

```
.content {
  grid-area: content;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

```
<div class="grid">
  <div class="title">Title</div>
  <div class="header">Header</div>
  <div class="sidebar">Sidebar</div>
  <div class="content">
    <div class="nestedGrid"></div>
    <div class="nestedGrid"></div>
    <div class="nestedGrid"></div>
  </div>
  <div class="footer">footer</div>
</div>
```

Instructor Demonstration

Nesting Grids


Break

Activity: CSS Grid, Part 2

In this activity, you'll keep working on the same file.

Suggested Time:

30 minutes

CSS

Time’s Up! Let’s Review.

Let's Review: CSS Grid, Part 2

- What does the property `align-content` do?
- What does the property `justify-content` do?
- Why would you want to nest a grid inside a grid?

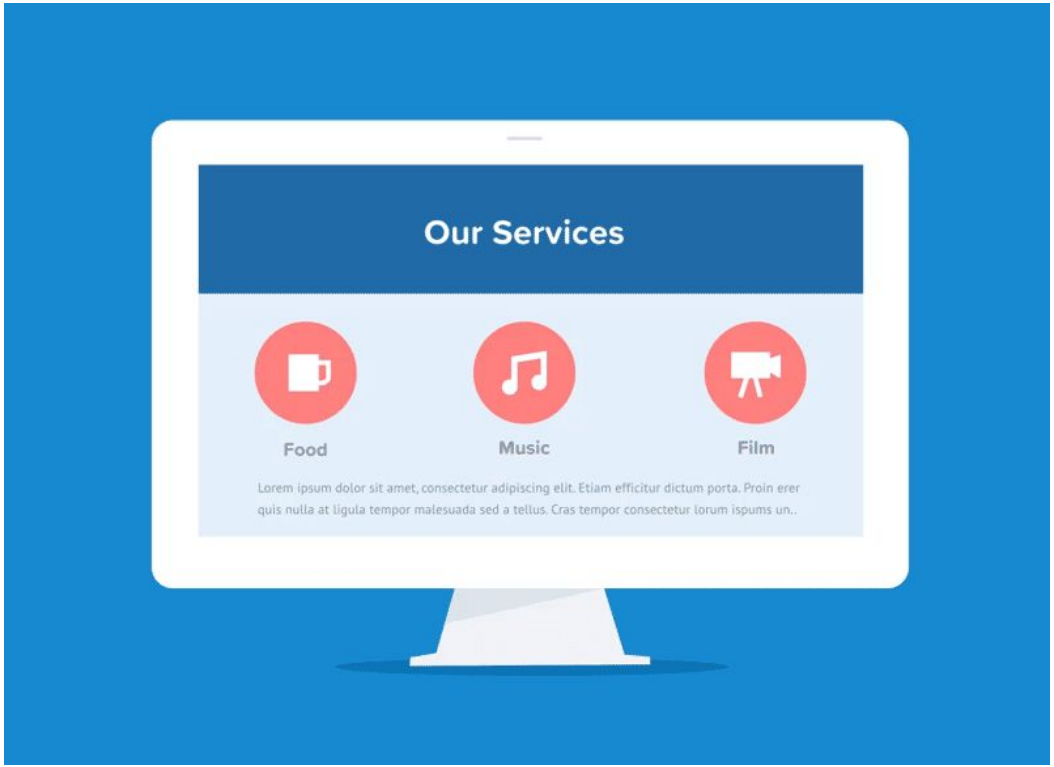
Responsive Web Design and Media Queries

Media queries define how CSS styles are applied in relation to the characteristics of the device viewport.

Media Queries

In layman's terms, media queries overwrite previously written CSS properties to alter our code to display correctly for different sizes.

This could be a smartphone, tablet, or even a projector. Websites need to respond correctly to all device sizes.



Media Queries

https://mediaqueri.es

Media Queries

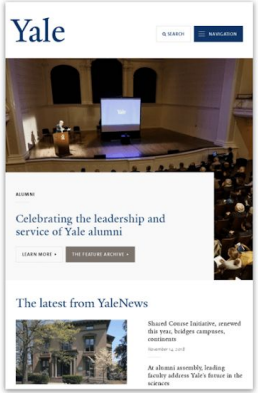
Mattermost
#1 Slack Alternative For Developers

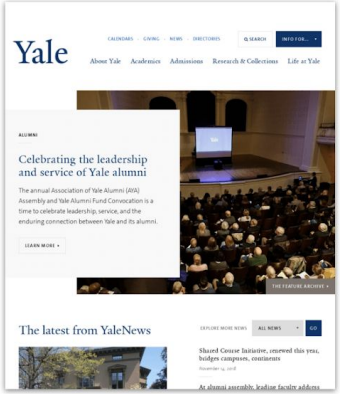
Messaging and workflow for developers. Write, deploy, and debug code as a team.

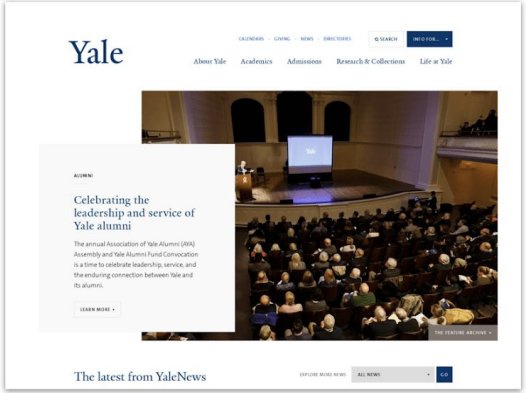
ads via Carbon

Yale

1





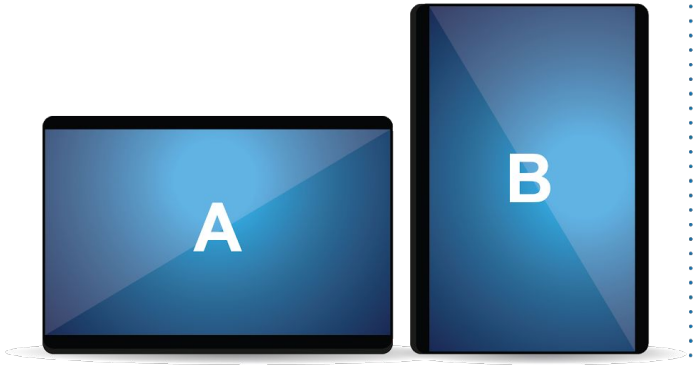


The **viewport** is the window of whatever device you are visiting the website on.

Viewport

(min-width: 800px)
(min-device-aspect-ratio: 1/1)

(min-width: 768px)
(min-device-width: 1024px)



(min-width: 320px)
(min-device-width: 480px)

↑ Designing mobile-friendly web-mapping applications using CSS3 media queries can provide an improved experience to your end users.

A `@media screen and (orientation: landscape)`

B `@media screen and (orientation: portrait)`

https://www.brownbagmarketing.com/blog/5-reasons-to-start-thinking-mobile-first/

Media Query Syntax

The `@media` selector specifies the start of our media query.

Here, we select what type of devices we are targeting. This one is targeting screens. You can also use:

- `all`
- `print`
- `screen`
- `speech`

We write new styles that will overwrite our old ones between the brackets.

```
@media only screen { }
```

Media Query Syntax

What about if we want to target specific widths? In this media query, we have two different statements.

The first is the viewport must be a `screen and`.

With `max-width`, this media query will be applied when the viewport is under 600px.

```
@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

Media Query Syntax

When this device's screen size is under 600px, the body's `background-color` will be changed to light blue.

```
@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

Media Query Syntax

How do they work?

Media queries overwrite previous properties you have specified in your CSS, but only when the viewport is the correct size.

This allows you to restructure your content however you see fit depending on the screen size.

```
body {
  background-color: red;
}

@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

What if I need to set up a media query between two specific sizes?

Media Query Syntax

This code is checking for a browser width between 1024px and 1280px.

There are many ways to write media queries, and we've only scratched the surface on how to write them.

```
@media all and (min-width: 1024px) and (max-width: 1280px) {
  body {
    background-color: green;
  }
}
```

Activity: CSS Grid: Part 3

In this activity, you'll keep working on the same file.

Suggested Time:

15 minutes

CSS

Time’s Up! Let’s Review.

Let's Review: Custom Breakpoints Review

Let's Review: Custom Breakpoints Review

01 Why do we care about grid layouts?

02 What are nested grids?

03 What is a breakpoint?

04 How do media queries work?


RECAP

Congratulations! Recap

Today, we learned:

01

CSS Grids

How to code a grid and use layout processes.

02

HTML nesting

How to think about making coded objects work together.

03

CSS positioning

How to move elements inside a grid container.

Questions?

the End